Solving of Eigenvalue and Singular Value Problems via Modified Householder Transformations on Shared Memory Parallel Computing Systems

Andrey Andreev and Vitaly Egunov(✉)

Volgograd State Technical University, Volgograd, Russia
andan2005@yandex.ru, vegunov@mail.ru

Abstract. Discusses the use of original modifications of Householder transformations for solving eigenvalue and singular value problems. Shared memory parallel computing systems are choosen as target computing systems. The proposed modifications allow to increase the computational performance due to the efficient use of cache memory and parallel execution of some transformation steps, which are traditionally performed serial. Mathematical descriptions of modified transformations are given, as well as software implementation issues and experimental results.

Keywords: Eigenvalues · Singular values · Householder transformation · Reflection transformation · Program performance · Cache memory · Shared memory system

1 Introduction

Finding eigenvalue and eigenvector systems, performing singular decomposition are widely used in solving a number of fundamental scientific and applied tasks. As an example, one can cite the problems of modeling various dynamic systems described by differential equations, specifically, when searching for the natural frequencies of oscillations of a dynamic system, the problem of determining the energy spectrum of quantum systems, image processing, etc.

The problems under consideration have a high computational complexity, which makes them the object of research for mathematicians and software engineers. The tasks themselves are well known. The task of finding its own decomposition is to find vectors and values that satisfy the condition (1).

$$Av = \lambda v \tag{1}$$

In this case, vector **v** is called the eigenvector of the linear operator A, the value λ is eigenvalue. In general, both λ and **v** can be complex even in the case of real A. Algorithms for solving the problem (1) can be divided into algorithms for finding all eigenvalues, as well as eigenvector systems, and algorithms for finding several (or probably the only one) eigenvalues. The most widely used are the iterative algorithms,

V. Voevodin and S. Sobolev (Eds.): RuSCDays 2019, CCIS 1129, pp. 131–151, 2019.
https://doi.org/10.1007/978-3-030-36592-9_12

that develop a sequence of transformations converging to the eigenvalues, as well as the development of a sequence of vectors converging to the eigenvectors of the linear operator.

$$A_{k+1} = U_k A_k U_k^* \tag{2}$$

Here U_k - unitary matrix, U_k^* - conjugate transpose matrix. In the case of real matrices, orthogonal and transposed matrices are used. All matrices A_i are similar, i.e. their eigenvalues are equal. There are no relatively simple direct algorithms for finding eigenvalues and eigenvector systems, however, such algorithms are known for matrices of a special kind. As an example, triangular matrices with eigenvalues are located on the main diagonal.

The problem of finding a singular value decomposition is to find vectors and values that satisfy the condition (3).

$$\mathrm{A} * \mathrm{u} = \lambda \mathrm{v} \tag{3}$$

Vectors u and v are called the left and right singular vectors corresponding to the singular value λ, A* is the matrix, conjugate transpose to A. The singular value decomposition of the matrix is the decomposition of the form (4).

$$A = U \Sigma V^* \tag{4}$$

Here A is the initial matrix, U and V areunitary matrices, Σ is the matrix which elements lying on the main diagonal are singular values. In the case of real matrices, orthogonal and transposed matrices are also used.

Most modern computing systems have a parallel architecture, as a result, currently there are many studies on the adaptation of known algorithms for use on parallel computing systems, as well as the study of the effectiveness of these algorithms when used on parallel systems with different characteristics, for example [1, 2]. In this work, we have optimized the modification of the known algorithms for solving the problems of finding a eigenvalues and singular values on parallel systems with shared memory, the result of these modifications reduces the time of solving these problems. The choice of this class of parallel systems is due to their wide prevalence. The choice of the object of research is determined by the demand for these tasks in solving various fundamental and applied problems. In many cases, these transformations are repeated many times, the calculations take a lot of time. Therefore, reducing the computation time seems to be an urgent task. One more reason for the paper is the fact that the authors have been engaged for several years in the study of issues related to the implementation of matrix transformations on heterogeneous computational systems, including the transformations mentioned above [3–6].

2 Used Methods

One of the methods of accelerating the process of calculating eigenvalues is to bring the original matrix of the general structure to the "almost triangular" structure, in which, in addition to the elements of the triangular matrix, the diagonal is stored under or above the main diagonal. Matrices of this type are called the upper or lower Hessenberg matrix, respectively. There is a finite sequence of transformations that leads a matrix of arbitrary structure to a Hessenberg structure with preservation of its eigenvalues. Such matrices, along with tridiagonal matrices, are the source for many eigenvalue search algorithms, such as, for example, the QR algorithm or the Jacobi method. Thus, it can be concluded that the reduction of the matrix of arbitrary structure to Hessenberg's one, and symmetric matrix – to the tridiagonal structure, is an integral part of a large number of algorithms for finding eigenvalues of the linear operator, and improving the efficiency of methods for solving this problem leads to a decrease in the time of solving the problem of finding eigenvalues as a whole.

In the case of singular value decomposition of matrices of General structure, one of the most effective methods is to reduce the matrix to a dual-diagonal structure by means of transformations (4), where in this case Σ is a dual-diagonal matrix, with further application of efficient algorithms for computing the singular value decomposition of a dual-diagonal matrix. Thus, improving the effectiveness of the methods of reduction of matrices to dual-diagonal structure leads to reduce the run-time of singular value decomposition in General.

Both (2) and (4) use unitary matrices as operators U and V, and orthogonal matrices in the case of real values computations. In this paper we consider the problem of reducing the matrix of General form to Hessenberg and dual-diagonal form in order to further calculate its eigenvalues or singular values. As a basic method will use Housholder reflections.

$$A = QRQ^T \tag{5}$$

Here A is initial matrix, R is Hessenberg or dual-diagonal matrix, Q is orthogonal matrix, representing a Housholder matrix multiplication, each matrix is uniquely determined by the reflection vector. In [4] examines the performance of the QR – decomposition by Householder reflections on parallel computing systems with shared memory. In particular, multiplication by the reflection matrix on the right (QR - decomposition) and on the left (LQ - decomposition) is considered. In the first case, the reflection vector is determined by the columns of the original matrix, the columns are also zeroed, in the second case, the reflection vector is determined by the rows of the original matrix, the rows are zeroed. The variant of LQ decomposition in this case is more preferable because of the lower probability of cache misses. A cache miss occurs when the microprocessor tries to access data that is not present in the cache memory. In this case, they must be loaded from the main memory. If the required data is in the cache, it is quickly retrieved. This event is called a cache hit. Improving the performance of the computer system is achieved when cache hits are implemented much more often than cache misses. The frequency of cache misses in this case can be

estimated as the probability of missing the required data in the cache memory during the next iteration of the transformation applied to the column or row of the matrix. As the data in the cache memory is loaded from the lines, the more effective from the point of view of usage of the cache memory are algorithms in which the processing is performed along the rows of the matrix. This algorithm is a variant of LQ – decomposition, in contrast to the variant of QR – decomposition, in which data processing is carried out along the columns of the matrix.

In the case of a two-way transformation (2) or (4), the situation is not so clear, because it is necessary to multiply on both sides to preserve the eigenvalues or singular values. Consider the reduction of the matrix of dimension n*n to the upper Hessenberg structure. To do this, you need to perform (n − 2) steps within each:

- calculate the elements of the reflection vector on the elements of the next column of the original matrix;
- multiply the original matrix by the reflection matrix on the right;
- multiply the original matrix by the reflection matrix on the left.

The algorithm for performing the k-th conversion step can be written as follows (6).

$$s_k = -sign\left(a_{k+1,k}\right)\left(\sum_{i=k+1}^{n} a_{ik}^2\right)^{\frac{1}{2}}, \qquad \varphi_k = \left(s_k^2 - s_k a_{k+1,k}\right)^{-1}$$

$$\boldsymbol{u}_k^T = \left(0, \ldots, 0, a_{k+1,k} - s_k, a_{k+2,k}, \ldots, a_{nk}\right)$$

$$\left.\begin{aligned} \lambda_j &= \varphi_k \boldsymbol{u}_k^{\boldsymbol{T}} \boldsymbol{a}_j \\ \boldsymbol{a}_j &= \boldsymbol{a}_j - \lambda_j \boldsymbol{u}_k \end{aligned}\right| j = \overline{k,n} \tag{6}$$

$$\left.\begin{aligned} \lambda'_j &= \varphi_k \boldsymbol{b}_j \boldsymbol{u}_k \\ \boldsymbol{b}_j &= \boldsymbol{b}_j - \lambda'_j \boldsymbol{u}_k^{\boldsymbol{T}} \end{aligned}\right| j = \overline{k,n}$$

Here $\boldsymbol{a}$ – columns of the original matrix, $\boldsymbol{b}$ – its rows, expression $j = \overline{k,n}$ indicates the range of indexes when processing rows and columns of the matrix in the current step of the transformation. The sequence of steps in this case is as follows:

- the reflection vector is determined from the values of the elements of the k-th column of the matrix under the main diagonal;
- multiplication by the Householder matrix on the right, which results in zero elements of k column, changes (n − 1) elements of all other columns, starting with (k + 1); the columns of the matrix can be processed in parallel;
- multiplication by the Householder matrix on the left, which changes (n − 1) elements of all matrix rows starting from k; the matrix rows can be processed in parallel.

To bring the matrix to a dual-diagonal structure, you need to perform more operations. This is due to the fact that the elements of the reflection vector are calculated at

both stages of the transformation. The algorithm of matrix reduction in the upper dual-diagonal structure can be written as follows (7).

$$s_k = -sign\left(a_{k,k}\right)\left(\sum_{i=k}^{n} a_{ik}^2\right)^{\frac{1}{2}}, \qquad \varphi_k = \left(s_k^2 - s_k a_{k,k}\right)^{-1}$$

$$\boldsymbol{u}_k^T = \left(0, \ldots, 0, a_{k,k} - s_k, a_{k+1,k}, \ldots, a_{nk}\right)$$

$$\left.\begin{array}{r} \lambda_j = \varphi_k \boldsymbol{u}_k^T \boldsymbol{a}_j \\ \boldsymbol{a}_j = \boldsymbol{a}_j - \lambda_j \boldsymbol{u}_k \end{array}\right| j = \overline{k, n} \tag{7}$$

$$\left.\begin{array}{r} s'_k = -sign\left(a_{k,k+1}\right)\left(\sum\limits_{i=k+1}^{n} a_{ki}^2\right)^{\frac{1}{2}}, \qquad \varphi'_k = \left(s_k'^2 - s'_k a_{k,k+1}\right)^{-1} \\ \boldsymbol{u}_k'^T = \left(0, \ldots, 0, a_{k,k+1} - s'_k, a_{k,k+2}, \ldots, a_{kn}\right) \\ \left.\begin{array}{r} \lambda'_j = \varphi_k \boldsymbol{b}_j \boldsymbol{u}'_k \\ \boldsymbol{b}_j = \boldsymbol{b}_j - \lambda'_j \boldsymbol{u}_k'^T \end{array}\right| j = \overline{k, n} \\ < n - 1 \end{array}\right\| k$$

The sequence of actions is generally similar to the sequence of actions when reduced to the upper Hessenberg structure, except:

- when multiplying on the right for column calculations $\boldsymbol{a}_j$ in the formation of the reflection vector are used the elements of the column under the main diagonal together with the elements of the main diagonal;
- when multiplying on the left generates a new reflection vector based on the row elements to the right of the main diagonal;
- the total number of steps in this case is (n − 1);
- when multiplying on the left is done fewer steps (n − 2), what the symbol indicates $k < n - 1$, indicates the number of steps within which you want to perform the conversions.

Since the transformations are generally similar, we focus on reducing the matrix to the Hessenberg structure, further extending the findings to the algorithm of reducing the matrix to the dual-diagonal form.

To reduce the matrix to the lower Hessenberg structure, perform the following steps at each step of the transformation (8).

$$s_k = -sign\left(a_{k,k+1}\right)\left(\sum_{i=k+1}^{n} a_{ki}^2\right)^{\frac{1}{2}}, \qquad \varphi_k = \left(s_k^2 - s_k a_{k,k+1}\right)^{-1}$$

$$\boldsymbol{u}_k^T = \left(0, \ldots, 0, a_{k,k+1} - s_k, a_{k,k+2}, \ldots, a_{kn}\right)$$

$$\left.\begin{aligned}\lambda'_j &= \varphi_k \boldsymbol{b_j u_k} \\ \boldsymbol{b_j} &= \boldsymbol{b_j} - \lambda'_j \boldsymbol{u_k^T}\end{aligned}\right| j = \overline{k,n} \qquad (8)$$

$$\left.\begin{aligned}\lambda_j &= \varphi_k \boldsymbol{u_k^T a_j} \\ \boldsymbol{a_j} &= \boldsymbol{a_j} - \lambda_j \boldsymbol{u_k}\end{aligned}\right| j = \overline{k,n}$$

In General, the same actions are performed, taking into account the following comments:

- reflection vectors are formed on the basis of matrix row elements;
- is transformed first row, and then columns of the matrices.

All three of the above transformations (6)–(8) use the natural data parallelism inherent in these methods to write parallel programs:

- when multiplying to the right, the matrix columns can be processed in parallel;
- when multiplying from the left, matrix rows can be processed in parallel.

A large number of publications are devoted to the analysis of these transformations. Much attention of researchers is attracted by the solving of the eigenvalues problem for matrices of a special kind, for example [7–9], the use of singular value decomposition in solving various technical problems [10]. They Transformation of Hessenberg matrices, including to obtain eigenvalue and singular value decompositions are investigated [11, 12]. Traditionally, the attention of researchers is attracted by the Householder reflections [1, 2, 12–16]. The main publications are devoted to the application of this transformation to solving various problems of scientific and applied nature, often on computer systems of a certain type. A number of works are devoted to the effective use of cache memory when performing matrix operations [17]. In this article, we propose original modifications of the Householder reflections, designed to write programs for parallel computing systems with shared memory, effectively using cache memory, thereby significantly speeding up calculations compared to traditional computing schemes.

3 Proposed Solutions

In this section, we consider the proposed modifications of the methods described above, designed to speed up calculations on parallel computing systems with shared memory. In both algorithms (6) and (8) of matrix reduction to Hessenberg structure, the same transformations are performed over the columns and rows of the matrix, only the order of actions is changed, so these steps on parallel computing systems will be performed at the same time. A significant difference of the presented algorithms is the scheme of reflection vector formation. And from this point of view, the algorithm of bringing the matrix to the lower Hessenberg structure is more preferable (8). This is due to the fact that the formation of the reflection vector is based on the elements of the matrix rows, in contrast to the algorithm (6), where the reflection vector is formed on

the basis of the column elements. The fact is that when using the elements of the columns of the matrix on parallel computing systems with shared memory will generate a greater number of cache misses. This effect has already been mentioned above. However, due to the fact that to determine the elements of the reflection vector in the total volume of calculations is small enough, both algorithms will have similar efficiency.

Consider these algorithms in terms of increasing their efficiency and reducing runtime on parallel systems with shared memory. There are several bottlenecks:

- in these algorithms, there are three serial stages at each step of the transformation, one of the stages is the calculation of the elements of the reflection vector, although there are implementations in which the elements of the reflection vector are pre-computed for the next step of the transformation, described, for example, in [4];
- when multiplied by the reflection matrix on the right, the columns of the matrix are processed, which significantly reduces the efficiency compared to the variant of calculations based on the processing of matrix rows.

First, we consider the modernization of these algorithms, aimed at improving the efficiency associated with the acceleration of processing columns of the original matrix.

In accordance with (6) in the process of reducing the original matrix to the upper Hessenberg structure when multiplied by the reflection matrix on the right, during which the next column of the original matrix is reset, the following actions are performed:

- the elements of the reflection vector are determined based on the values of the corresponding column of the original matrix;
- new values of matrix columns are calculated; columns can be processed in parallel, because in this case there are no information dependencies.

In this case, the following calculations are performed during the recalculation of the column values:

- the values of the inner products of the columns and the reflection vector are determined;
- new values of column elements are calculated based on the obtained values of inner products.

Both data stages are well parallelized, but due to the fact that the data is processed along the columns, a sufficiently large value of cache misses is generated, which greatly slows down the calculation process.

To eliminate the negative impact of cache misses, the following algorithm for calculating the new values of the matrix columns is proposed:

- at the first stage all necessary inner products are calculated;
- at the second stage, the new values of the column elements are calculated, but since the values of all inner products are known at this stage, the processing can be carried out line by line, for each element of the row, use its own value of the inner product.

This process can be written as follows (9). Here and in the future, the reflection vector formation algorithm is similar to (6) when reduced to the upper Hessenberg structure and similar to (8) when reduced to the lower Hessenberg structure. Therefore, except where it is really necessary, it will not be cited.

$$\begin{gathered}
\boldsymbol{sc} = \varphi_k \boldsymbol{u}_k^T \boldsymbol{A} \\
\boldsymbol{D} = diag(sc_0, sc_1, \ldots, sc_n) \\
\boldsymbol{U}[:j] = [\boldsymbol{u_k}] \\
\boldsymbol{SCU} = \boldsymbol{UD} \\
\boldsymbol{b_j} = \boldsymbol{b_j} - \boldsymbol{scu_j} \big| j = \overline{k,n} \\
\left. \begin{gathered} \lambda'_j = \varphi_k \boldsymbol{b_j} \boldsymbol{u_k} \\ \boldsymbol{b_j} = \boldsymbol{b_j} - \lambda'_j \boldsymbol{u_k^T} \end{gathered} \right| j = \overline{k,n}
\end{gathered} \tag{9}$$

Before processing the columns, the vector of inner products of *sc* is calculated as the product of the reflection vector u_k^T on the original matrix *A* taking into account the scale factor φ_k. Next is calculation of new values of columns of the matrix, and the computation is carried out row by row, which is much more efficient from the point of view of using the cache memory. It should be noted that (9) uses the following symbols: *D* – a diagonal matrix containing the values of inner products *sc* on its main diagonal, *U* **–** a matrix whose columns are composed of a reflection vector, *scu* – rows of the matrix *SCU*, *b* - rows of the matrix A. As can be seen from (9), when multiplying the matrix both on the right and on the left, the processing is carried out row by row, which significantly increases the efficiency of the program implementation of the method, reducing the execution time of the program.

An algorithm for reducing the matrix to the lower Hessenberg structure using a row-oriented scheme of processing columns of the matrix is given in (10).

$$\begin{gathered}
\left. \begin{gathered} \lambda'_j = \varphi_k \boldsymbol{b_j} \boldsymbol{u_k} \\ \boldsymbol{b_j} = \boldsymbol{b_j} - \lambda'_j \boldsymbol{u_k^T} \end{gathered} \right| j = \overline{k,n} \\
\boldsymbol{sc} = \varphi_k \boldsymbol{u}_k^T \boldsymbol{A} \\
\boldsymbol{D} = diag(sc_0, sc_1, \ldots, sc_n) \\
\boldsymbol{U}[:j] = [\boldsymbol{u_k}] \\
\boldsymbol{SCU} = \boldsymbol{UD} \\
\boldsymbol{b_j} = \boldsymbol{b_j} - \boldsymbol{scu_j} \big| j = \overline{k,n}
\end{gathered} \tag{10}$$

In (9) and (10) it is not shown the formation of the reflection vector, which is carried out similarly to (6) and (7), respectively, i.e. on the basis of columns and rows of the matrix, respectively.

Let us now consider the modernization of the computational process associated with the preliminary calculation of the reflection vector. In the basic version of the algorithm, as well as in the modifications discussed above, the elements of the reflection vector are formed in the serial part of the program. According to Amdahl's Law, it is serial computing that limits the acceleration of a program on parallel computing systems. Accordingly, one of the ways to improve the efficiency of programs, reduce the time of their execution, can be considered a reduction in the share of serial calculations in the total volume of calculations.

The idea is based on the transfer of the stage of determining the elements of the reflection vector from the serial part of the calculations to the parallel one. After analyzing the algorithm, we can conclude that in order to form the reflection vector for the step number (k + 1) it is not necessary to wait for the end of all calculations associated with step number k. For a one-way transformation, this means that when you perform step k of the algorithm to bring the matrix to the upper triangular structure by multiplying the reflection matrix on the right, the reflection vector can be formed after the elements of the column number (k + 1) are formed. Accordingly, when performing the step with number k of the matrix reduction algorithm to the lower triangular structure by multiplying the reflection matrix on the left, the reflection vector can be formed after the formation of the elements of the row with number (k + 1). With parallel implementation of the algorithm, this leads to some acceleration of calculations, as the share of serial calculations in the total volume of calculations decreases.

When performing a two-way transformation, this strategy does not work, because when multiplying by the reflection matrix on the other hand, the elements on the basis of which the reflection vector is formed will change. Thus, when reduced to the upper Hessenberg structure, it would be erroneous to form a reflection vector based on the values of the column obtained at the next step when multiplying on the right, since the values of the elements of this column will change when multiplied by the reflection matrix on the left. The same situation will be observed when the matrix is reduced to the lower Hessenberg structure. In both cases, the reflection vector can be generated only after the second stage of the transformation is completed in this step. It looks like applying reflection transformations to bring the matrix to Hessenberg or dual-diagonal structure.

Consider a transformation algorithm that includes a preliminary computation of the reflection vector elements used to bring the matrix to the upper Hessenberg structure. The formation of the elements of the reflection vector in this case is based on the columns of the matrix, however, the final stage of the transformation at each step includes the processing of rows. Obviously, all elements of the target column will be known only after the whole step is completed. However, it is obvious that when

processing the next row, the value of the next element of the target column becomes known, which is used to form the reflection vector in the next step. Thus, it is possible to form a reflection vector element by element, removing this stage from the serial part of the program, increasing its efficiency.

The formal definition of the process of bringing the matrix to the upper Hessenberg structure with a preliminary calculation of the elements of the reflection vector can be written as follows (11).

$$s_{k|j+1} = 0$$

$$\left.\begin{aligned} \lambda_j &= \varphi_k \boldsymbol{u}_k^T \boldsymbol{a}_j \\ \boldsymbol{a}_j &= \boldsymbol{a}_j - \lambda_j \boldsymbol{u}_k \end{aligned}\right| j = \overline{k,n}$$

$$\left.\begin{aligned} \lambda'_j &= \varphi_k \boldsymbol{b}_j \boldsymbol{u}_k \\ \boldsymbol{b}_j &= \boldsymbol{b}_j - \lambda'_j \boldsymbol{u}_k^T \\ \boldsymbol{u}_{k+1}^T[j] &= \boldsymbol{b}_j[k+1] \big| j > k+2 \\ s_{k|j+1} &= s_{k|j+1} + \boldsymbol{b}_j[k+1]^2 \big| j \geq k+2 \end{aligned}\right| j = \overline{k,n} \tag{11}$$

$$s_{k|j+1} = -sign\left(a_{k+2,k+1}\right) s_{k|j+1}^{\frac{1}{2}},$$

$$\varphi_{k|j+1} = \left(s_{k|j+1}^2 - s_{k|j+1} a_{k+2,k+1}\right)^{-1}$$

$$\boldsymbol{u}_{k+1}^T[0:(\mathrm{k}+1)] = 0$$

$$\boldsymbol{u}_{k+1}^T[\mathrm{k}+2] = a_{k+2,k+1} - s_{k|j+1}$$

During the second stage of the next step of the transformation, the elements of the reflection vector are formed in the parallel part of the program, as well as the accumulation of the module $s_{k|j+1}$, necessary to calculate the scaling factor $\varphi_{k|j+1}$. A subscript indicates that this value will be used in step k to process column number j + 1. It should be noted that it was not possible to completely abandon the serial part of the calculations, because some compensation of the value of the module $s_{k|j+1}$ after processing all the rows is required. The code that would implement the algorithm given in (11) would look like this. This code snippet is written in the C programming language using OpenMP technology.

```
#pragma omp parallel for
    for(int row = step; row < N; row++)
    {
            double scalar = 0;
            for(int i = (step + shift); i < N; i++)
                    scalar += matr[row * N + i] * vect[i];
            scalar *= gamma;
            for(int i = (step + shift); i < N; i++)
            {
                    matr[row * N + i] -= scalar * vect[i];

                    if (i == (step + shift))
                    {
                            if (row > (step + 1 + shift))
                                    vectAdd[row] = matr[row * N + i];
                            if (row >= (step + 1 + shift))
#pragma omp atomic
                                    s += matr[row * N + i] * matr[row * N +
i];
                    }
            }
    }
    s = sqrt(s);
    if (matr[(step + 1 + shift)  * N + step + 1] > 0)
            s = -s;
    gamma = 1 / (s * s - matr[(step + 1 + shift) * N + step + 1] * s);
    for (int i = 0; i < (step + 1 + shift); i++)
            vectAdd[i] = 0;
    vectAdd[step + 1 + shift] = matr[(step + 1 + shift) * N + (step + 1)] - s;
```

Despite the fact that the reflection vector is formed almost entirely in the parallel part of the program, this code is inefficient because of the large number of checks carried out in each thread when processing each row. In addition, #pragma omp atomic synchronization can reduce the efficiency. The following code changes are made to improve efficiency:

- unroll the loop by performing its first iteration outside the loop; there is no need to check if (i == (step + shift));
- remove the check if (row > (step + 1 + shift)); the reflection vector will be corrected in the serial part of the program, zeroing the necessary elements;
- remove the check if (row > = (step + 1 + shift)); the value of the module is adjusted in the serial part of the program, subtracting from it the squares of unnecessary elements;
- remove #pragma omp atomic synchronization, each thread accumulates a module in its variable, their values are summed in the serial part of the program.

Get the following code.

```
#pragma omp parallel for
        for(int row = step; row < N; row++)
        {
                double scalar = 0;
                for(int i = (step + shift); i < N; i++)
                        scalar += matr[row * N + i] * vect[i];
                scalar *= gamma;
                matr[row * N + step + shift] -= scalar * vect[step + shift];
                vectAdd[row] = matr[row * N + step + shift];
                sAr[numT] += matr[row * N + step + shift] * matr[row * N + step +
shift];
                for(int i = (step + shift + 1); i < N; i++)
                        matr[row * N + i] -= scalar * vect[i];

        }
        s = 0;
        for(int ii = 0; ii < nT; ii++)
                s += sAr[ii];
        s -= matr[step * N + step + shift] * matr[step * N + step + shift];
        if (shift > 0)
                s -= matr[(step + shift) * N + step + shift] * matr[(step + shift) * N +
step + shift];
        s = sqrt(s);
        if (matr[(step + 1 + shift)  * N + step + 1] > 0)
                s = -s;
        gamma = 1 / (s * s - matr[(step + 1 + shift) * N + step + 1] * s);
        for (int i = 0; i < (step + 1 + shift); i++)
                vectAdd[i] = 0;
        vectAdd[step + 1 + shift] = matr[(step + 1 + shift) * N + (step + 1)] - s;
```

The shift parameter is assumed to be 1 when performing a two-way transformation to convert the matrix to Hessenberg structure and is assumed to be 0 when performing a one-way transformation to convert the matrix to triangular form. It is worth noting that before you start the transformation, you must calculate the elements of the reflection vector used to perform the first step of the transformation.

The transformation in this case will take the form (12).

$$s_{k|j+1} = 0$$

$$\left.\begin{aligned} \lambda_j &= \varphi_k \boldsymbol{u}_k^T \boldsymbol{a}_j \\ \boldsymbol{a}_j &= \boldsymbol{a}_j - \lambda_j \boldsymbol{u}_k \end{aligned}\right| j = \overline{k,n} \tag{12}$$

$$\left.\begin{aligned}\lambda'_j &= \varphi_k \boldsymbol{b_j} \boldsymbol{u_k} \\ \boldsymbol{b_j} &= \boldsymbol{b_j} - \lambda'_j \boldsymbol{u_k^T} \\ \boldsymbol{u_{k+1}^T}[j] &= \boldsymbol{b_j}[k+1] \\ s_{k|j+1} &= s_{k|j+1} + \boldsymbol{b_j}[k+1]^2 \end{aligned}\right| j = \overline{k,n}$$

$$s_{k|j+1} = s_{k|j+1} - \mathbf{A}[k,k+1]^2 - \mathbf{A}[k+1,k+1]^2$$

$$s_{k|j+1} = -sign\left(a_{k+2,k+1}\right) s_{k|j+1}^{\frac{1}{2}},$$

$$\varphi_{k|j+1} = \left(s_{k|j+1}^2 - s_{k|j+1} a_{k+2,k+1}\right)^{-1}$$

$$\boldsymbol{u_{k+1}^T}[0:(\mathrm{k}+1)] = 0$$

$$\boldsymbol{u_{k+1}^T}[\mathrm{k}+2] = a_{k+2,k+1} - s_{k|j+1}$$

The reduction of the matrix to the lower Hessenberg structure with pre-calculated elements of the reflection vector will be as follows (13).

$$s_{k|j+1} = 0$$

$$\left.\begin{aligned}\lambda'_j &= \varphi_k \boldsymbol{b_j} \boldsymbol{u_k} \\ \boldsymbol{b_j} &= \boldsymbol{b_j} - \lambda'_j \boldsymbol{u_k^T} \end{aligned}\right| j = \overline{k,n} \tag{13}$$

$$\left.\begin{aligned}\lambda_j &= \varphi_k \boldsymbol{u_k^T} \boldsymbol{a_j} \\ \boldsymbol{a_j} &= \boldsymbol{a_j} - \lambda_j \boldsymbol{u_k} \\ \boldsymbol{u_{k+1}^T}[j] &= \boldsymbol{a_j}[k+1] \\ s_{k|j+1} &= s_{k|j+1} + \boldsymbol{a_j}[k+1]^2 \end{aligned}\right| j = \overline{k,n}$$

$$s_{k|j+1} = s_{k|j+1} - \mathbf{A}[k+1,k]^2 - \mathbf{A}[k+1,k+1]^2$$

$$s_{k|j+1} = -sign\left(a_{k+1,k+2}\right) s_{k|j+1}^{\frac{1}{2}},$$

$$\varphi_{k|j+1} = \left(s_{k|j+1}^2 - s_{k|j+1} a_{k+1,k+2}\right)^{-1}$$

$$\boldsymbol{u_k^T}[0:(\mathrm{k}+1)] = 0$$

$$\boldsymbol{u_k^T}[\mathrm{k}+2] = a_{k+1,k+2} - s_{k|j+1}$$

Finally, both approaches can be combined, obtaining an effective implementation of the Householder transformation of to bring the matrix to Hessenberg structure in the framework of solving the eigenvalues problem.

The modification of the Householder reflection algorithm, consisting in the row-oriented scheme of processing of the matrix and the preliminary calculation of the elements of the reflection vector, applied to the problem of reducing the matrix to the upper Hessenberg structure, will be as follows (14).

$$s_{k|j+1} = 0$$

$$\boldsymbol{sc} = \varphi_k \boldsymbol{u}_k^T \boldsymbol{A}$$

$$\boldsymbol{D} = diag(sc_0, sc_1, \ldots, sc_n)$$

$$\boldsymbol{U}[:j] = [\boldsymbol{u_k}]$$

$$\boldsymbol{SCU} = \boldsymbol{UD}$$

$$\boldsymbol{b_j} = \boldsymbol{b_j} - \boldsymbol{scu_j} \big| j = \overline{k,n}$$

$$\left.\begin{aligned} \lambda'_j &= \varphi_k \boldsymbol{b_j} \boldsymbol{u_k} \\ \boldsymbol{b_j} &= \boldsymbol{b_j} - \lambda'_j \boldsymbol{u_k^T} \\ \boldsymbol{u_{k+1}^T}[j] &= \boldsymbol{b_j}[k+1] \\ s_{k|j+1} &= s_{k|j+1} + \boldsymbol{b_j}[k+1]^2 \end{aligned}\right| j = \overline{k,n} \tag{14}$$

$$s_{k|j+1} = s_{k|j+1} - \mathbf{A}[k,k+1]^2 - \mathbf{A}[k+1,k+1]^2$$

$$s_{k|j+1} = -sign\left(a_{k+2,k+1}\right) s_{k|j+1}^{\frac{1}{2}},$$

$$\varphi_{k|j+1} = \left(s_{k|j+1}^2 - s_{k|j+1} a_{k+2,k+1}\right)^{-1}$$

$$\boldsymbol{u_{k+1}^T}[0:(\mathrm{k}+1)] = 0$$

$$\boldsymbol{u_{k+1}^T}[\mathrm{k}+2] = a_{k+2,k+1} - s_{k|j+1}$$

(14) does not show the formation of the reflection vector for the first step of the transformation. Similarly, it is possible to obtain an algorithm for reducing the matrix to the lower Hessenberg structure, however, it will not give a tangible increase in efficiency compared to the algorithm (10), since in this case the elements of the reflection vector are determined in accordance with the elements of the matrix, i.e. quite effectively.

The resulting modification of the Householder reflection algorithm can be extended to an algorithm to bring the matrix to dual-diagonal form. Moreover, a significant increase in efficiency will be achieved both in the case of bringing the matrix to the upper dual - diagonal and to the lower dual - diagonal structure. This is due to the fact that in these algorithms, at each step, the calculation of the elements of the reflection vector is carried out twice – when multiplying on the right by the elements of the columns, when multiplying on the left by the elements of the rows. In (15), an algorithm for reducing the matrix to the upper dual – diagonal structure, taking into account the proposed modifications, in (16) - to the lower dual - diagonal structure is given.

$$
\begin{gathered}
s_{k|j+1} = 0 \\
\boldsymbol{sc} = \varphi_k \boldsymbol{u}_k^T \boldsymbol{A} \\
\boldsymbol{D} = diag(sc_0, sc_1, \ldots, sc_n) \\
\boldsymbol{U}[:j] = [\boldsymbol{u_k}] \\
\boldsymbol{SCU} = \boldsymbol{UD} \\
\boldsymbol{b_j} = \boldsymbol{b_j} - \boldsymbol{scu_j} \big| j = \overline{k,n}
\end{gathered}
\tag{15}
$$

$$
\left.
\begin{gathered}
s'_k = -sign\left(a_{k,k+1}\right)\left(\sum_{i=k+1}^{n} a_{ki}^2\right)^{\frac{1}{2}}, \\
\varphi'_k = \left(s'^2_k - s'_k a_{k,k+1}\right)^{-1} \\
\boldsymbol{u}'^T_k = \left(0, \ldots, 0, a_{k,k+1} - s'_k, a_{k,k+2}, \ldots, a_{kn}\right) \\
\left.
\begin{gathered}
\lambda'_j = \varphi_k \boldsymbol{b_j} \boldsymbol{u'_k} \\
\boldsymbol{b_j} = \boldsymbol{b_j} - \lambda'_j \boldsymbol{u'^T_k} \\
\boldsymbol{u^T_{k+1}}[j] = \boldsymbol{b_j}[k+1] \\
s_{k|j+1} = s_{k|j+1} + \boldsymbol{b_j}[k+1]^2
\end{gathered}
\right| j = \overline{k,n} \\
s_{k|j+1} = s_{k|j+1} - \mathbf{A}[k, k+1]^2 \\
s_{k|j+1} = -sign\left(a_{k+1,k+1}\right) s^{\frac{1}{2}}_{k|j+1}, \\
\varphi_{k|j+1} = \left(s^2_{k|j+1} - s_{k|j+1} a_{k+1,k+1}\right)^{-1} \\
\boldsymbol{u^T_{k+1}}[0:\mathrm{k})] = 0 \\
\boldsymbol{u^T_{k+1}}[\mathrm{k}+1] = a_{k+2,k+1} - s_{k|j+1}
\end{gathered}
\right| k < n-1
$$

$$s'_k = -sign\left(a_{k,k}\right)\left(\sum_{i=k}^{n} a_{ki}^2\right)^{\frac{1}{2}}, \qquad \varphi'_k = \left(s'^2_k - s'_k a_{k,k}\right)^{-1}$$

$$\boldsymbol{u}'^T_k = \left(0, \ldots, 0, a_{k,k} - s'_k, a_{k,k+1}, \ldots, a_{kn}\right)$$

$$s_k = 0$$

$$\left.\begin{aligned} \lambda'_j &= \varphi_k \boldsymbol{b_j} \boldsymbol{u'_k} \\ \boldsymbol{b_j} &= \boldsymbol{b_j} - \lambda'_j \boldsymbol{u'^T_k} \\ \boldsymbol{u^T_{k+1}}[j] &= \boldsymbol{b_j}[k] \\ s_k &= s_k + \boldsymbol{b_j}[k]^2 \end{aligned}\right| j = \overline{k, n}$$

$$s_k = s_k - \mathbf{A}[k,k]^2$$

$$s_k = -sign\left(a_{k+1,k}\right)s_k^{\frac{1}{2}}, \qquad \varphi_k = \left(s_k^2 - s_k a_{k+1,k}\right)^{-1}$$

$$\boldsymbol{u^T_k}[0:\mathrm{k}] = 0 \tag{16}$$

$$\boldsymbol{u^T_k}[\mathrm{k}+1] = a_{k+1,k} - s_k$$

$$\left.\begin{aligned} sc &= \varphi_k \boldsymbol{u}_k^T A \\ D &= diag(sc_0, sc_1, \ldots, sc_n) \\ U[:j] &= [\boldsymbol{u_k}] \\ SCU &= UD \\ \boldsymbol{b_j} &= \boldsymbol{b_j} - \boldsymbol{scu_j} \end{aligned}\right\| \begin{aligned} &\\ &\\ &\\ &\\ &j = \overline{k, n} \end{aligned} \quad \Big\| \; k < n-1$$

In (15) and (16), the calculation of the reflection vector based on the row elements can also be made in the parallel part of the program, however, this will not give a tangible increase in efficiency.

4 Results of Computational Experiments

The object of research was a parallel computer system with shared memory, built on the basis of multi-core microprocessors Xeon E5-2650v3(x2) 2.3 GHz. This system is a part of the computer cluster of Volgograd State Technical University.

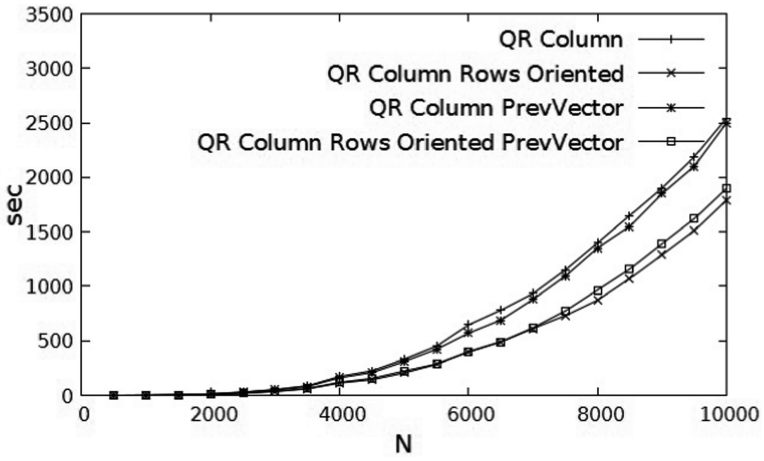

Fig. 1. Time of operation of serial variants of programs of matrix reduction to the upper Hessenberg structure.

Figure 1 shows the results of serial programs to bring matrices of arbitrary form to the upper Hessenberg structure. Programs were developed in the C programming language, the real type with double precision was used.

In Fig. 1, the following symbols are used:

- QR Column – traditional algorithm based on the classical Householder transformation (6);
- QR Column Rows Oriented – the algorithm based on the proposed row – oriented modifications of the Householder transformation (9);
- QR Column PrevVector– the algorithm based on the proposed modification of the Householder transformation with a preliminary calculation of the elements of the vector reflection (11);
- QR Column Rows Oriented PrevVector– an algorithm based on the simultaneous application of both proposed modifications of the Householder transformation (14).
- Analyzing the results presented in Fig. 1, we can draw the following conclusions:
- the least efficient is the implementation of the classical Householder transformation (6);
- despite the fact that the option with a preliminary calculation of the elements of the reflection vector (11) has a somewhat greater operating complexity of its serial software implementation is somewhat more rapid implementation of the classical transformation (6); the gain in execution time is about 5–6%; this is because in this case more effectively interacts with the cache memory in the formation of the elements of the vector reflection;
- the fastest is the variant with the use of row-oriented modification (9);
- the use of both modifications (14) works a little longer due to the greater operational complexity; at the same time, the gain due to the effective formation of the reflection vector elements in this case becomes insufficient, since the row – oriented scheme itself is effective in terms of interaction with cache memory.

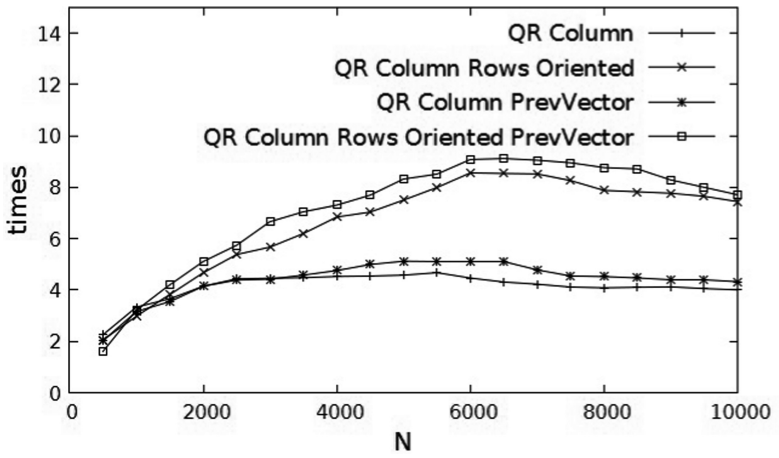

Fig. 2. Acceleration values of parallel variants of programs to bring the matrix to the upper Hessenberg structure in relation to the basic algorithm (6), five parallel threads.

Figure 2 shows the values of acceleration parallel implementations of the same algorithms against the baseline algorithm (6) – a serial algorithm based on the classical Householder transformation. The simulation was carried out on 5, 10 and 15 parallel threads. Figure 2 shows the simulation results on five threads.

In this case, the best option is the one in which both modifications were used:

- row-oriented computing optimize the operation of the microprocessor with cache memory;
- a preliminary calculation of the elements of a vector of reflection in the parallel implementations reduces their share of serial calculations, reducing the time of work of the program's General.

Analyzing the results presented in Fig. 2, we can conclude that all the assumptions made during the development of algorithms (9), (11) and (14) were justified and successfully confirmed by computational experiments. On 5 parallel threads, the operation of the basic version of the algorithm was accelerated by more than 9 times. In the resultant acceleration contributes not only to parallelize computations on multiple cores of the microprocessor, but also optimize the interaction with the cache memory.

Figure 3 shows the values of acceleration for parallel implementations of the same algorithms on ten parallel threads.

It can be observed from the Fig. 3, that the proposed modifications of the basic algorithm still have the best acceleration indicators, but their efficiency has decreased with the increase of the number of parallel threads.

When the matrix is reduced to a dual-diagonal structure while solving the problem of singular value decomposition, the same pattern is observed as in Figs. 1, 2 and 3.

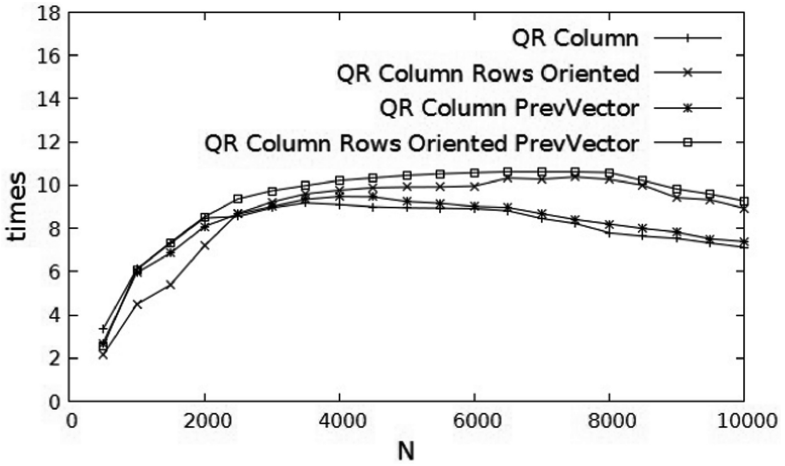

Fig. 3. Acceleration values of parallel programs to bring the matrix to the upper Hessenberg structure in relation to the basic algorithm (6), ten parallel threads.

5 Summary

According to the results of computational experiments we can conclude that the proposed modification of the custom Householder reflection transformations as applied to the solution of problems of a eigenvalues and singular values decompositions proved to be quite efficient on parallel computational systems with shared memory. It should also be noted that the proposed algorithms accelerate calculations not only for parallel but also for serial implementations. It can be stated that in the case of serial implementation it is advisable to use the proposed row-oriented reflection transformation, in the case of parallel – row-oriented transformation with the preliminary calculation of the elements of the reflection vector, i.e. the option in which both proposed modifications are used. The reduction of program execution time is achieved primarily due to the effective organization of work with cache memory, which significantly reduces the frequency of cache misses and as a consequence significantly speeds up the calculation process. Moreover, in the parallel implementations an additional increase in acceleration of calculations is achieved due to the reduction of the share of serial calculations. The results presented in this paper are original and obtained for the first time.

Acknowledgements. Work is performed with the financial support of the Russian Foundation for Basic Research - project#18-47-340010 r_a and the financial support of the Administration of Volgograd region.

References

1. Merchant, F., Vatwani, T., Chattopadhyay, A., Raha, S., Nandy, S.K., Narayan, R.: Efficient realization of householder transform through algorithm-architecture co-design for acceleration of QR Factorization. IEEE Trans. Parallel Distributed Syst. **29**(8), 1707–1720 (2018)
2. Tomas Dominguez, A.E., Quintana Orti, E.S.: Fast blocking of householder reflectors on graphics processors. In: Proceedings - 26th Euromicro International Conference on Parallel, Distributed, and Network-Based Processing, PDP 2018, pp. 385–393 (2018)
3. Andreev, A., Doukhnitch, E., Egunov, V., Zharikov, D., Shapovalov, O., Artuh, S.: Evaluation of hardware implementations of CORDIC-like algorithms in FPGA using OpenCL kernels. In: Kravets, A., Shcherbakov, M., Kultsova, M., Iijima, T. (eds.) JCKBSE 2014. CCIS, vol. 466, pp. 228–242. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-11854-3_20
4. Egunov, V.A.: Implementation of QR and LQ decompositions on shared memory parallel computing systems. In: Egunov, V.A., Andreev, A.E. (eds.) 2016 2nd International Conference on Industrial Engineering, Applications and Manufacturing (ICIEAM), Chelyabinsk, Russia, 19–20 May 2016, 5 p. IEEE (2016)https://doi.org/10.1109/icieam.2016.7911607
5. Getmanskiy, V., Andreev, A.E., Alekseev, S., Gorobtsov, A.S., Egunov, V., Kharkov, E.: Optimization and parallelization of CAE software stress-strain solver for heterogeneous computing hardware. In: Kravets, A., Shcherbakov, M., Kultsova, M., Groumpos, P. (eds) CIT&DS 2017. CCIS, vol 754, pp. 562–674. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-65551-2_41
6. Glinsky, B., et al.: The co-design of astrophysical code for massively parallel supercomputers. In: Carretero, J., et al. (eds.) ICA3PP 2016. LNCS, vol. 10049, pp. 342–353. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-49956-7_27
7. Tian, Y.: Some results on the eigenvalue problem for a fractional elliptic equation. Boundary Value Problems **1**, 13 (2019)
8. Baker, C.G., Hetmaniuk, U.L., Lehoucq, R.B., Thornquist, H.K.: Anasazi software for the numerical solution of large-scale eigenvalue problems. ACM Trans. Math. Softw. **36**(3), art. no. 13 (2009). https://doi.org/10.1145/1527286.1527287
9. Polizzi, E.: Density-matrix-based algorithm for solving eigenvalue problems. Phys. Rev. B - Condensed Matter Mat. Phys. **79**(11), art. no. 115112 (2009). https://doi.org/10.1103/physrevb.79.115112
10. Bogoya, J.M., Grudsky, S.M., Malysheva, I.S.: Extreme individual eigenvalues for a class of large hessenberg toeplitz matrices. Operator Theory Adv. Appl. **271**, 119–143 (2018)
11. Vatankhah, S.: Large-scale inversion of magnetic data using golub-kahan bidiagonalization with truncated generalized cross validation for regularization parameter estimation. J. Earth Space Phys. **44**(4), 29–39 (2019)
12. Salam, A., Kahla, H.B.: An upper J-Hessenberg reduction of a matrix through symplectic Householder transformations. Computers and Mathematics with Applications (2019)
13. Liu, G., Liu, Y., Guo, M., Li, P., Li, M.: Variational inference with Gaussian mixture model and householder flow. Neural Networks **109**, 43–55 (2019)
14. Li, S., Cao, G., Wei, S.: Improved measurement matrix and reconstruction algorithm for compressed sensing. In: Proceedings of 2018 IEEE 8th International Conference on Electronics Information and Emergency Communication, ICEIEC 2018, 8473512, pp. 136–139 (2018)

15. Noble, J.H., Lubasch, M., Stevens, J., Jentschura, U.D.: Diagonalization of complex symmetric matrices: generalized Householder reflections, iterative deflation and implicit shifts. Comput. Phys. Commun. **221**, 304–316 (2017)
16. Bujanovic, Z., Karlsson, L., Kressner, D.: A householder-based algorithm for hessenberg-triangular reduction. SIAM J. Matrix Anal. Appl. **39**(3), 1270–1294 (2018)
17. Eljammaly, M., Karlsson, L., Kågström, B.: On the Tunability of a New Hessenberg Reduction Algorithm Using Parallel Cache Assignment. In: Wyrzykowski, R., Dongarra, J., Deelman, E., Karczewski, K. (eds.) PPAM 2017. LNCS, vol. 10777, pp. 579–589. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-78024-5_50